Faster Code....

Faster

Intel® Parallel Studio XE 2017

Support for 2nd Gen Intel® Xeon Phi™ Processors

September 2016
Create Faster Code...Faster

Intel® Parallel Studio XE

- High Performance Scalable Code
  - C++, C*, Fortran*, Python* and Java*
  - Standards-driven parallel models: OpenMP*, MPI, and Intel® Threading Building Blocks (Intel® TBB)

- New for 2017
  - 2nd generation Intel® Xeon Phi™ processor and Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
    - Optimized compilers and libraries
    - Vectorization and threading optimization tools
    - High bandwidth memory optimization tools
  - Faster Python application performance
  - Faster deep learning on Intel® architecture

http://intel.ly/perf-tools
## Intel® Parallel Studio XE

Tools to build, analyze and scale high performance software

### BUILD

**COMPOSER EDITION**

- **Optimizing Compilers**
  - Intel® C/C++ and Fortran Compilers
- **Machine Learning and Analytics Library**
  - Intel® Data Analytics Acceleration Library
- **Fast Math Library**
  - Intel® Math Kernel Library

**PROFESSIONAL EDITION**

- **Performance Scripting**
  - Intel® Distribution for Python*
- **Image, Signal, and Compression Routines**
  - Intel® Integrated Performance Primitives
- **Task-Based Parallel C++ Template Library**
  - Intel® Threading Building Blocks

### ANALYZE

**PROFESSIONAL EDITION**

- **Performance Profiler**
  - Intel® VTune™ Amplifier
- **Vectorization Optimization & Thread Design**
  - Intel® Advisor

**MEMORY AND THREADING DEBUGGING**

- **Memory and Threading Debugging**
  - Intel® Inspector

### SCALE

**CLUSTER EDITION**

- **MPI Profiler**
  - Intel® Trace Analyzer and Collector
- **Scalable Cluster Messaging**
  - Intel® MPI Library

**CLUSTER DIAGNOSTIC EXPERT SYSTEM**

- **Cluster Diagnostic Expert System**
  - Intel® Cluster Checker

---

*Other names and brands may be claimed as the property of others.*
What’s New: Intel® Compilers 2017
AVX-512, easier vectorization, faster coarrays, standards

Vector and parallel programming improvements

- AVX2 and AVX-512 for new processors (e.g., 2nd gen. Intel® Xeon Phi™ processor) [C++* & Fortran*]
- Enhanced optimization/vectorization reports for easier code modernization [C++ & Fortran]
- SIMD Data Layout Template to facilitate vectorization for your C++ code [C++]
- Virtual function vectorization capability [C++]
- Faster coarrays – up to twice as fast as 16.0 on non-trivial coarray Fortran programs [Fortran]

Standards

- Full C11 and C++14, initial C++17 support
- Almost complete Fortran 2008 support, further interoperability with C (part of draft Fortran 2015)
- OpenMP* 4.5 support
SIMD Data Layout Template
Improve Productivity and Boost C++ Performance

- Quickly convert “Array of Structures” to “Structure of Arrays” representation.
- Increase productivity: Use predefined templates with minimal effort, and let SDLT do the vectorization for you.
- Improve performance: SDLT vectorizes your code by making memory access contiguous, which can lead to more efficient code and better performance.
- Seamless integration: SDLT follows the familiar Intel vector programming model.

"We used SDLT to vectorize the deformer code in Premo, the in-house animation tool for DreamWorks Animation. The performance improvements we were able to achieve were dramatic, and these improvements will translate directly into higher quality characters that will be seen on-screen in future movies. Also the library itself was easy to use and integrate into our existing codebase."

Martin Watt
Principal Engineer,
DreamWorks Animation
Boost Application Performance on Windows* and Linux*
Intel® C++ and Fortran Compilers

Boost C++ application performance on Windows* & Linux* using Intel® C++ Compiler
(higher is better)

Boost Fortran application performance on Windows* & Linux* using Intel® Fortran Compiler
(higher is better)

Floating Point

Integer

Windows
Estimated SPECfp®_rate_base2006

Linux
Estimated SPECint®_rate_base2006

Windows
Linux

PGI* 15.10
Visual C++
2015
Intel C++
17.0
Clang®
GCC*: 6.1.0
Clang®
GCC*: 6.1.0

PGI* 15.10
Visual C++
2015
Intel C++
17.0
Clang®
GCC*: 6.1.0
Clang®
GCC*: 6.1.0

1.13
1.10
1.08
1.00
1.39
1.03
1.28
1.71
1.02
1.55

White Papers
Technical Briefs
Code Examples
Intel® C++ Compiler
Intel Fortran Compiler

Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240)
Linux OS: Red Hat Enterprise Linux Server release 7.1 (Maipo), kernel 3.10.0-229.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240)

Relative geomean performance, SPEC* benchmark - higher is better

C++ Configuration: Windows hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, HT enabled, TB enabled, 32 GB RAM; Linux hardware: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 256 GB RAM, HyperThreading is on. Software: Intel compilers 17.0, Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x86/x64, GCC 6.1.0, PGI 15.10, Clang/LLVM 3.8
Linux OS: Red Hat Enterprise Linux Server release 7.1 (Maipo), kernel 3.10.0-229.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240)

Fortran Configuration: Hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, Hyperthreading enabled, TB enabled, 32 GB RAM. Software: Intel Fortran compiler 170, Absoft*15.0.1, PGI Fortran* 15.10 (Windows)/16.4 (Linux), Open64* 4.5.2, gfortran* 6.1.0, Linux OS: Red Hat Enterprise Linux Server release 7.2, Kernel 3.10.0-327.45.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240).

Relative geomean performance, Polyhedron* benchmark– higher is better

Windows
Linux

PGI Fortran* 15.10
Absolt* 15.0.1
Open64* 4.5.2
PGI Fortran* 16.4
Absolt* 15.0.1
Intel Fortran 17.0
Intel Fortran 17.0

1.00
1.29
1.00
1.00
1.14
1.26
1.43
1.87

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSMark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.
Highlights: Intel® Distribution for Python* 2017
Focus on advancing python performance closer to native speeds

Easy, out-of-the-box access to high-performance Python
- Prebuilt, accelerated Distribution for numerical and scientific computing, data analytics, HPC. Optimized for Intel® architecture.
- Drop-in replacement for your existing Python. No code changes required.

Drive performance with multiple optimization techniques
- Data analytics with pyDAAL, enhanced thread scheduling with Intel® Threading Building Blocks, Jupyter* notebook interface, Numba*, Cython*.
- Scale easily with optimized mpi4py and Jupyter notebooks.

Faster access to latest optimizations for Intel® architecture
- Distribution and individual optimized packages available through Conda* and Anaconda Cloud*.
- Optimizations upstreamed back to main Python* trunk.

*Other names and brands may be claimed as the property of others.
Intel® Distribution for Python* 2017
Near-native performance speedups on Intel® architecture

Python* Performance as a Percentage of C/Intel® MKL for Intel® Xeon Phi™ Product Family (Higher is Better)

- LAPACK 1 core
- LAPACK 64 cores
- FFT 1 core
- FFT 64 cores

See this slide for configurations

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.
What’s New: Intel® Math Kernel Library 2017
Optimized for the latest processors and deep learning

- Intel® Math Kernel Library (Intel® MKL) 2017 optimized for 2nd generation Intel® Xeon Phi™ processors
- Improved optimizations for the latest Intel® processors
- Optimized math functions for deep learning neural networks (CNN and DNN)
- Improved ScaLAPACK performance for symmetric eigensolvers on clusters
- New data fitting functions based on B-splines and monotonic splines
- Extended Intel® TBB threading layer support for all BLAS level-1 functions
Intel® MKL: Performance Benefit to Applications

DGEMM Performance
On Intel® Xeon Phi™ Processor 7250

The latest version of Intel® MKL unleashes the performance benefits of Intel® architectures.
What’s New: Intel® Data Analytics Acceleration Library (Intel® DAAL) 2017

- Neural Networks
- Python* API (a.k.a. PyDAAL)
  - Easy installation through Anaconda or pip
- New data source connector for KDB+
- Open source project on GitHub*

Fork me on GitHub: https://github.com/01org/daal
Example Performance: Intel® DAAL vs. Spark* MLLib

Intel® DAAL vs. Spark* MLLib
K-means Performance Comparison on Eight-node Cluster

**Configuration Info:** Versions: Intel® Data Analytics Acceleration Library 2017, Spark 1.2; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86_64.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit [http://www.intel.com/performance](http://www.intel.com/performance).
What’s New: Intel® Threading Building Blocks 2017
Excellent scalability on 2\textsuperscript{nd} generation Intel® Xeon Phi™ processors

static\_partitioner class
\begin{itemize}
  \item Helps minimizing overhead of parallel loops
\end{itemize}

streaming\_node class
\begin{itemize}
  \item Enables heterogeneous streaming computations within the flow graph.
\end{itemize}

Added method to isolate execution of a group of tasks or an algorithm from other tasks submitted to the scheduler. A preview feature for 2017.

Python* module is added to replace Python's thread pool class.

Graph/stereo example is added.

Improvements to graph/fgbzip example (added async\_msg usage example)
Excellent Performance Scalability with Intel® Threading Building Blocks 2017 on Intel® Xeon Phi® Processor

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.
What’s New: Intel® Integrated Performance Primitives (Intel® IPP) 2017

- Extended optimization for AVX-512, Intel® Xeon® processors and 2nd generation Intel® Xeon Phi™ processors and coprocessors
- Intel® IPP Platform-Aware APIs in the image and signal processing domains are added to support external threading and 64-bit data length
- Significantly improved performance of zlib compression functions is
- Extension of IPP optimized functionality in OpenCV
- Limited pre-silicon optimizations for next generation Intel Xeon Phi processors and CNL EP/XE server
Intel® Integrated Performance Primitives
Faster performance on the Intel® Xeon Phi™ Processor

Intel® IPP Signal Processing Functions Performance Boost
Intel® IPP AVX-512 Optimization Code vs. Compiled C Code

Configurations:
- SW Versions: Intel® Integrated Performance Primitives (Intel® IPP) 2017, RHEL Server 7.0, 64-bit
- Hardware: Intel® Xeon Phi™ Processor 7250, 32 MB L2 cache, 1.4 GHz, Single-threaded
- Compiler: Intel C++ Compiler 16.0 with -march=avx512

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.
Analysis Tools for Intel® Xeon Phi™ Processors
Profile, vectorize and debug modern code

Intel® VTune™ Amplifier – performance profiler
- Performance profiling
- Memory analysis for high bandwidth memory
- Threading optimization analysis

Intel® Advisor – vectorization optimization
- Measure FLOPs and optimize for AVX-512
- Loop-carried dependency analysis
- Memory access pattern & footprint analysis

Intel® Inspector – thread & memory debugger
- Debug non-deterministic threading errors
- Find & debug memory errors
Intel® VTune™ Amplifier Tuning
Four critical optimizations for Intel® Xeon Phi™ Processors

1) High Bandwidth Memory
   - Decide which data structures to place in MCDRAM
   - See performance problems by memory hierarchy
   - Measure DRAM and MCDRAM bandwidth

2) Scalability of MPI and OpenMP
   - Serial vs. Parallel time
   - Imbalance, overhead cost, parallel loop parameters

3) Micro Architecture Efficiency
   - See the efficiency of your code in the core pipeline
   - Zero in on details with custom PMU events

4) Vectorization Efficiency – Use Intel® Advisor
   - Optimize for AVX-512 with or without AVX-512 hardware
Vectorize and Thread for Performance Boost
Threaded and Vectorized can be much faster than either one alone

**The Difference Is Growing With Each New Generation of Hardware**

- **2007**: X5472
- **2009**: X5570
- **2010**: X5680
- **2012**: E5-2600
- **2013**: E5-2600 v2
- **2014**: E5-2600 v3
- **2016**: E5-2600 v4

**Analyzing Software** and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit [http://www.intel.com/performance](http://www.intel.com/performance).

**Optimization Notice**
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.*
“Automatic” Vectorization Often Not Enough
A good compiler can still benefit greatly from vectorization optimization

Compiler will not always vectorize
- Check for Loop Carried Dependencies using **Intel® Advisor**
- All clear? Force vectorization.
  C++ use: pragma simd, Fortran use: SIMD directive

Not all vectorization is efficient vectorization
- Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with **Intel® Advisor**.
- Consider data layout changes
  **Intel® SIMD Data Layout Templates** can help

The benchmarks on the previous slides did not all “auto vectorize”. Compiler directives were used to force vectorization and get more performance.

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the **Intel® SIMD Data Layout Templates** (Intel® SDLT) to map data into a more efficient layout for vectorization.
Next Gen Intel® Xeon Phi™ Support

Vectorization Advisor runs on, and optimizes for, the Intel® Xeon Phi™ processor

AVX-512 ERI – specific to Intel® Xeon Phi™ processor

Efficiency (72%), Speed-up (11.5x), Vector Length (16)

Performance optimization problem and advice how to fix it

Program metrics
Elapsed Time: 142.79s
Vector Instruction Set: AVX, AVX2, AVX512, SSE, SSE2
Number of CPU Threads: 4

Loop metrics
Total CPU time 454.08s
Time in 88 vectorized loops 41.86s 9.2%
**Tuning for AVX-512 without AVX-512 Hardware**

**Intel® Advisor - Vectorization Advisor**

Use –axCOMMON-AVX512 –xAVX compiler flags to generate both code-paths

- AVX2 code path (executed on Haswell and earlier processors)
- AVX512 code path for newer hardware

Compare AVX and AVX-512 code with Intel® Advisor

<table>
<thead>
<tr>
<th>Loops</th>
<th>Self Time</th>
<th>Loop Type</th>
<th>Vectorized Loops</th>
<th>Instruction Set Analysis</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in s352_at_loopstr.cpl5933]</td>
<td>0.641s1</td>
<td>Vectorized (Body)</td>
<td>AVX2</td>
<td>INSERTS: FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s352_at_loopstr.cpp9383]</td>
<td>n/a</td>
<td>Vectorized (Body)</td>
<td>AVX2</td>
<td>GATHERS: FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s352_at_loopstr.cpp9383]</td>
<td>n/a</td>
<td>Vectorized (Body) [Not Executed]</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>0.406s1</td>
<td>Vectorized Versions</td>
<td>AVX2</td>
<td>FMA; NT-stores</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>n/a</td>
<td>Vectorized (Body)</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>n/a</td>
<td>Vectorized (Body) [Not Executed]</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>0.45s1</td>
<td>Vectorized (Body)</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>n/a</td>
<td>Vectorized (Remainder)</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
<tr>
<td>[loop in s125_AompSparalle_for...</td>
<td>n/a</td>
<td>Vectorized (Remainder) [Not Executed]</td>
<td>AVX2</td>
<td>FMA</td>
<td></td>
</tr>
</tbody>
</table>

**Speed-up estimate:**

- 13.5x (AVX2) vs. 30.6x (AVX512)
Debug Difficult Non-Deterministic Threading Errors

Intel® Inspector 2017 – Memory and thread debugger

Runs Native on Intel® Xeon Phi™ Processors

- Simplifies workflow for Intel® Xeon Phi™ processor development
- Tip: Reduce thread count to ≤ 30 for best Intel Xeon Phi processor performance while running Intel Inspector

New C++ Language Features

- Full C++ 11 support including std::mutex and std::atomic

New Easier Identification of Threading Bugs

- Variable name causing error is shown (global, static & stack) in addition to the code lines
Intel® MPI Library: Scalable Cluster Messaging

Optimized MPI application performance

- Application-specific tuning
- Automatic tuning
- New! - Support for 2nd gen Intel® Xeon Phi™ processor
- New! - Support for Intel® Omni-Path Architecture Fabric

Lower-latency and multi-vendor interoperability

- Industry leading latency
- Performance optimized support for the fabric capabilities through OpenFabrics*(OFI)

Faster MPI communication

- Optimized collectives

Sustainable scalability up to 340K cores

- Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements

More robust MPI applications

- Seamless interoperability with Intel® Trace Analyzer and Collector
What’s New: Intel® MPI Library 2017

- Optimized for 2nd gen. Intel® Xeon Phi™ processors
  - Usage of specially optimized memcpy
  - Tuning of shared memory collectives on single nodes
- Support for Intel® Omni-Path Architecture fabric
- General optimization of RMA
- General optimization and speed up startup time and MPI tune utility
Intel® MPI Library
Superior performance on Intel® Xeon Phi™ processors
Intel® Trace Analyzer and Collector

MPI Profiling

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Idealizer

Automatically detect performance issues and their impact on runtime
MPI Performance Snapshot
Scalable Profiling for MPI and Hybrid

**Lightweight**: Low overhead profiling for 100K+ ranks.

**Scalable**: Performance variation at scale can be detected sooner.

**Identify Key Metrics**: Shows MPI/OpenMP* imbalances.
What’s New: Intel® Trace Analyzer and Collector

- Support for 2nd gen. Intel® Xeon Phi™ processors
- Improved scalability of imbalance profiler by up to 10X
- Improved MPI Snapshot feature HTML output
An **expert system** approach that provides cluster systems expertise in a tool

- Verifies system health
- Offers suggested actions
- Provides extensible framework
- API for integrated support
Additional Material
Intel® Parallel Studio XE 2017

- **Product page** – Overview, features, FAQs, support...
- **Product brief** – Overview of the three versions and features
- **Training materials** – Videos, tech briefs, documentation...
- **Evaluation guides** – Step-by-step walk through
- **Reviews**

Additional Development Products:

- **Intel® Software Development Products**
Problem Size and Configuration Information
Intel® Distribution for Python* Benchmarks

<table>
<thead>
<tr>
<th>Hardware/Problem Size</th>
<th>dot</th>
<th>lu</th>
<th>det</th>
<th>inv</th>
<th>cholesky</th>
<th>fft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon® processor (32 core) and Intel® Xeon Phi™ processor (64 core)</td>
<td>(20k, 10k) and (10k, 20k)</td>
<td>(35k, 35k)</td>
<td>(15k, 15k)</td>
<td>(25k, 25k)</td>
<td>(40k, 40k)</td>
<td>520k</td>
</tr>
<tr>
<td>Intel Xeon processor (1 core)</td>
<td>(20k, 5k) and (5,20k)</td>
<td>(20k, 20k)</td>
<td></td>
<td>(10k, 10k)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Xeon Phi processor (1 core)</td>
<td>(20k, 300) and (300, 20k)</td>
<td>(6k, 6k)</td>
<td>(4k, 4k)</td>
<td>(2k, 2k)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Configuration Info:** apt/atlas: installed with apt-get, Ubuntu* 16.10, Python* 3.5.2, numpy* 1.11.0, scipy* 0.17.0; pip*/openblas*: installed with pip, Ubuntu 16.10, python 3.5.2, numpy 1.11.1, scipy 0.18.0; Intel Python: Intel® Distribution for Python 2017; Hardware: Intel Xeon processor: Intel Xeon processor E5-2698 v3 @ 2.30 GHz (2 sockets, 16 cores each, HT=off), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Intel Xeon Phi processor: Intel® Xeon Phi™ processor 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz
## Configurations for 2007-2016 Benchmarks

### Platform Hardware and Software Configuration

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unscaled Core Frequency</th>
<th>Cores/ Socket</th>
<th>Num Sockets</th>
<th>L1 Data Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>Memory</th>
<th>Memory Frequency</th>
<th>Memory Access</th>
<th>H/W Prefetchers Enabled</th>
<th>HT Enabled</th>
<th>Turbo Enabled</th>
<th>C States</th>
<th>O/S Name</th>
<th>Operating System</th>
<th>Compiler Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon™ 5472 Processor</td>
<td>3.0 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>6 MB</td>
<td>None</td>
<td>32 GB</td>
<td>800 MHz</td>
<td>UMA</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5570 Processor</td>
<td>2.9 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>8 MB</td>
<td>48 GB</td>
<td>1333 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5680 Processor</td>
<td>3.33 GHZ</td>
<td>6</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>12 MB</td>
<td>48 MB</td>
<td>1333 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2690 Processor</td>
<td>2.9 GHZ</td>
<td>8</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>20 MB</td>
<td>64 GB</td>
<td>1600 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2697v2 Processor</td>
<td>2.7 GHZ</td>
<td>12</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>30 MB</td>
<td>64 GB</td>
<td>1867 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>RHEL 7.1</td>
<td>3.10.0-229.el7.x86_64</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2600v3 Processor</td>
<td>2.2 GHZ</td>
<td>18</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>46 MB</td>
<td>128 GB</td>
<td>2133 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.13.5-202.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2600v4 Processor</td>
<td>2.3 GHZ</td>
<td>18</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>46 MB</td>
<td>256 GB</td>
<td>2400 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>RHEL 7.0</td>
<td>3.10.0-123.el7.x86_64</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2600v4 Processor</td>
<td>2.2 GHZ</td>
<td>22</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>56 MB</td>
<td>128 GB</td>
<td>2133 MHz</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>CentOS 7.2</td>
<td>3.10.0-327.el7.x86_64</td>
<td>icc version 14.0.1</td>
</tr>
</tbody>
</table>

### Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Performance measured in Intel Labs by Intel employees.
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804